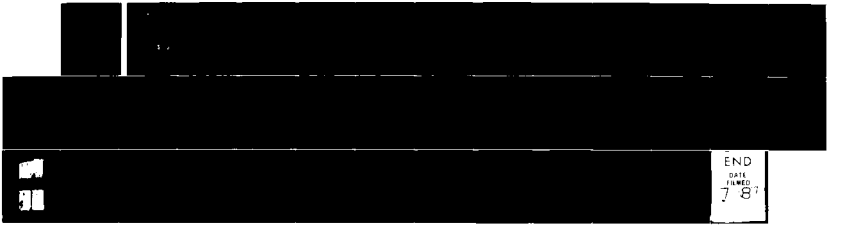
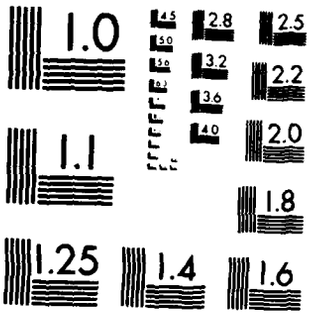
AD-A181 311 A MULTI-PROCESSOR SLAVE PERIPHERAL CONTROLLER(U) 1/1
AERONAUTICAL RESEARCH LABS MELBOURNE (AUSTRALIA)
J F HARVEY NOV 86 ARL-AERO-TM-385

UNCLASSIFIED F/G 12/6 NL



END
DATE FILMED
7-87



(12)

ARL-AERO-TM-385

AR-004-507

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

AERONAUTICAL RESEARCH LABORATORIES

MELBOURNE, VICTORIA

Aerodynamics Technical Memorandum 385

A MULTI-PROCESSOR SLAVE PERIPHERAL CONTROLLER

by

J.F. Harvey

NOVEMBER 1986

AR-004-507

DEPARTMENT OF DEFENCE
DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION
AERONAUTICAL RESEARCH LABORATORIES

Aerodynamics Technical Memorandum 385

A MULTI - PROCESSOR SLAVE PERIPHERAL CONTROLLER

by

J.F. Harvey

SUMMARY

A system providing multiple slave co-processors operating independently, yet in parallel upon the same VME bus, all under the control of a master processor is described.

Data are passed unidirectionally from master to slave enabling rapid processing of complex algorithms for the control of external peripherals.

POSTAL ADDRESS: Director, Aeronautical Research Laboratories,
P.O. Box 4331, Melbourne, Victoria, 3001, Australia.

CONTENTS

1. INTRODUCTION

An intelligent peripheral controller is described. The controller operates as a slave processor in a multiple processor environment.

The system utilizes Motorola MC68000 microprocessors operating on VME bus (Reference 1) and Eurocard construction. Each slave processor is totally self contained with individual bus buffering, memory and input/output.

Multiple slave processors connect to a common VME bus operating under the control of a master microprocessor. The master may operate under its own resident program or under the direct control of an external host computer. Passing of programs (algorithms) and exchange of data to the slave are controlled by the master processor. However, once the slaves are set up they operate independently of each other and the master until again interrupted. Each slave, although capable of independent operation, may operate as a synchronous or asynchronous system under hand shake control with the master or other slaves.

The system was originally designed to control high speed (10,000 step/second) stepper motors to position a six degrees of freedom, hot wire anemometer probe. The application for the probe was in flow measurement around a model in either the low speed or transonic wind tunnels at the Aeronautical Research Laboratories (ARL).

Both hardware and software for the system is described. The memorandum concludes with suggestions concerning alternative uses for the system.

Reference 1 VMEBUS Specification Manual Motorola MVMEBS/D1 1982

2. DESIGN PHILOSOPHY

Before describing the operation of the system in detail, it is useful to set out the general design philosophy.

(i) The system is controllable from a host computer via a RS232C serial interface.

(ii) Each slave processor operates independently upon complex positioning algorithms contained within their memory.

(iii) Synchronous and asynchronous hand-shake control between each slave is provided.

(iv) Speed of operation is limited by the driven devices and not the software execution time.

(v) Developmental software is down-line loadable from the host to the master and transferrable from master to a selected slave for performance evaluation.

3. HARDWARE

3.1. General

The Motorola MC68000 microprocessor is a high performance 16 bit data bus processor with 16 registers of 32 bit width, capable of directly addressing 16 Mega bytes of memory. This processor, with VME bus and Eurocard hardware, has been selected by ARL as a preferred system for in-house electronic instrumentation.

A variety of cards have been designed and constructed at ARL to provide building blocks for data acquisition and control systems.

The master processor section of the system requires a minimum of three cards. These are a MC68000L8 processor, a combined RAM/ROM memory and a dual serial RS232C interface (Figure 1). The slave processors were required to fit the same chassis hardware, so are VME bus compatable Eurocards. The amount of circuitry required for each slave was too great to fit onto a single Eurocard. Therefore, a piggy-back arrangement of two cards (Fig. 2) was selected and is known as a Slave Peripheral Controller (SPC). The main card contains a MC 68000 ZB10 microprocessor, the VME bus interface, the watch-dog timer, the independent slave system clock, memory and input/output address decoding (Fig. 3). The piggy-backed card contains 8 kbytes of RAM, 8kbytes of ROM, the Parallel Interface Timer (PIT MC68230-10) and the output buffering circuit.

An ARL designed 19 slot VME backplane accepts eight SPC's and three cards for the main processor (Fig. 1). Each SPC occupies two card slots of the VME back plane. If additional master processor cards are required, such as extra memory or an arithmetic processor the number of SPC's to a 19 slot VME back plane must be reduced.

For control of the six degrees of freedom hot wire anemometer probe the number of SPC's is six, thus permitting a maximum of seven master processor cards to be connected to the back plane. Smaller or larger systems may be assembled depending upon the task requirements, electrical loading and mechanical constraints of the VME bus system.

A listing of all ARL drawing numbers associated with the SPC board is provided in Table 1.

3.2. Bus Communication Master to Slave

There is no direct interconnection between the VME bus and the internal SPC bus. All communication between the two bus's is achieved by data transfers via back-to-back latch's (Fig. 3).

The two processor systems (master & slave) employ interrupts to initiate an exchange of status communication codes via a back-to-back status latch. The status latch is four bits wide and the status codes are therefore limited to sixteen (Fig. 4). To achieve control for the exchange of data from master memory to slave memory and the reverse, as well as to initiate the execution of a program residing in the slave memory, requires seven status codes.

Level 3 (IRQ3) user interrupt vector is called by the SPC onto the VME bus. All SPC's are hardwired to this one interrupt level. To identify which SPC in a multiple system requires service, a different interrupt vector number is returned by each SPC. This 8 bit hexidecimal number is preset by links on the SPC cards. Hence up to 256 individual SPC's may connect to the master processor, if the electrical loading specifications of the VME bus are met and interconnections are practical.

The master processor, on being interrupted on level 3 by the slave processor, completes execution of its current instruction then accesses the interrupt by asserting interrupt acknowledge and reading back from the slave the interrupt vector number. This vector number is automatically multiplied by 4 by the master to obtain the address of the interrupt service routine from the MC68000 Exception Vector Assignment (Fig. 5). A slave initiates an interrupt by writing bit 5 via its SPC data bus into the interrupt latch.

The master may also interrupt the slave by writing bit 5 via the VME data bus to the other half of the back-to-back interrupt latch. This calls a SPC level 4 interrupt autovector. In calling an autovectored interrupt the processor does not access the bus for an interrupt vector number, but accepts the Exception Vector Assignment at vector number 28 as the address of the interrupt service routine.

The interrupt latch is an extension of the status latch and is addressed differently from the back-to-back data transfer latch (Fig. 6). The status latch is at an odd address because it is only one byte wide (8 bits) and is decoded by the appropriate address and the Lower Data Strobe (LDS). Whereas, the data latch is at an even address and is one word wide (16 bits). This is decoded by the appropriate address and both the Lower (LDS) and Upper Data Strobes (UDS).

The address decoders for both the data latch and the status latch on the VME bus are programmed into Programmable Array Logic (PAL) circuits. Likewise the address decoders for the data and status latch's on the SPC bus are programmed in PAL's. The address decoding on the VME bus side is different for each SPC (Fig. 6), whereas, the address decoding on each individual SPC bus is identical (Fig. 7). This has the advantage that the software data handling routines are common within each SPC.

3.3. SPC Processor Circuitry

3.3.1. Processor

A Motorola MC68000ZB10 microprocessor was selected as the processor for the SPC. This is a 68000 packaged in a JEDEC type B leadless chip carrier, running at a clock frequency of 10 MHz.

The leadless chip carrier uses less board space, being physically one third the size of the dual-in-line 64 pin package used as the master processor. The higher clock speed of 10 MHz (compared with 8 mHz for the master) ensures that the processor instruction cycle is minimal, enabling complex positioning algorithms to be executed within the high speed step rate of 10,000 steps/second of the stepper motors.

3.3.2. Watch Dog Timer

A 6 bit shift register, enabled each time an SPC access address is asserted, provides a 6 microsecond delay before initiating a bus error. This ensures that the system will not hang-up should the handshake signal DTACK not be returned.

The memory is relatively slow with an inherent DTACK delay of 300 nanoseconds, whereas, the Parallel Interface and Timer (PIT) returns DTACK almost immediately. Likewise, the response of DTACK from the data and status latch is immediate.

Should a fault occur and bus error is asserted, the slave processor forces an exception and enters the bus error routine (Vector number 2 of the Exception Vector Assignment) from which recovery can be obtained and an error flag set for the master.

3.3.3. Memory

Four by twenty eight pin sockets are provided for the memory integrated circuits. This is configured so that two ROM's either 2732A (4K bytes), 2764 (8Kbytes) or 27128 (16Kbytes) of erasable read only memory and two RAM's either 6116 (2Kbytes) or 6264 (8 Kbytes) of read/write memory may be accommodated on

the piggy-backed board. The maximum memory capability is therefore 48 Kbytes, made up of program 32Kbytes (2 x 27128) and read/write memory 16 Kbytes (2 x 6264).

The various types of memory chips are link selectable on the piggy-backed board. This memory is arranged to be word wide (16 bits), hence the need for two chips of each type, allowing faster access to both data and program than would byte access.

3.3.4. PIT

The Parallel Interface and Timer (PIT) is a programmable input/output interface, designed to be compatible with 68000 microprocessors. The parallel interface operates in unidirectional or bi-directional modes either 8 or 16 bits wide. Also included within the PIT is a 5 bit prescaler and a 24 bit wide counter timer. This timer can generate periodic interrupts, a square wave output or a single interrupt after a programmed time period.

Four hand-shake lines are available on the PIT, which enable external devices to interrupt the SPC or provide synchronous or asynchronous operation of two or more SPC's within a system.

The PIT is interconnected for possible use of all the facilities available within the device. The multi-function lines of Port C may be utilized, via a series of hardwired links, for Port C to perform input/output functions or act as a timer and interrupt handling device.

3.3.5. SPC Interrupts

Various interrupts may be employed on the SPC processor. As discussed in Section 3.2, the VME bus master calls a level 4 interrupt autovector. When this level is called VPA is asserted informing the processor that this is an autovector and not a user vector interrupt (Reference 2). User vector interrupts are also employed on the SPC. Level 1 (IRQ1) may be called by the PIT timer TOUT pin active. Level 2 (IRQ2) may be called by the PIT programmed to the handshake lines H1 to H4 (Reference 3). The PIT provides five separate interrupt vectors, the vector number to be returned to the processor is pre-loaded into special interrupt vector registers.

3.3.6. External I/O Buffering

The PIT outputs (Port A to C) are unable to sink current of any consequence and board space for electrical buffering is provided. All Port A to C lines and the 4 hand-shake lines are connected to stakes. Also, the 40 pin ribbon connector header (to external devices) is connected to stakes. Buffering circuits may be wire-wrapped between the stakes or the PIT connected directly to the 40 pin ribbon header.

The DC electrical characteristics of the PIT outputs are:

Output Voltage	Load Current
VOH = 2.4V	ILH = - 100 A
VOL = 0.5V	ILO = 2.4 MA

Reference 2 Motorola ADI-814-R5 MC68000 March 1985
Reference 3 Motorola ADI-860-R2 MC68230 Dec 1983

3.3.7. SPC System Reset

The SPC is provided with a power-on-reset line which is held low for several milliseconds at switch "on" of the +5 volt VCC.

To regain control, in the advent of a fault condition, the main VME bus reset is common to all SPC resets. Therefore, an initialization switch may be utilized for manual overall system reset.

3.3.8. SPC Clock

Both the MC68000ZB10 and the MC682300-10 (PIT) are 10 MHz dynamic refresh devices and a crystal controlled clock module of 10 MHz is provided. This clock may also be connected to the Timer IN pin (TIN) for use with the 5 bit prescaler and the 24 bit counter timer function.

4. SOFTWARE

4.1. General

The memory map for the master processor is different from that allocated to the slaves. The master map has a much larger memory capacity (ROM capacity of the master is larger than the combined RAM/ROM capacity of a slave). Therefore, memory transfers from master to slave and vise versa requires an offset address. The offset chosen for the hot wire anemometer probe application is hex 10000. This means that data or program residing in the master memory at hex 12000, when transferred to the slave, resides at hex 2000. The offset also applies for the reverse transfer of data from the slave at hex 2000 to the master at hex 12000. The offset is

built into the software as a constant and may be changed depending upon the application.

In a minimum system, three routines are required to control master and slave.

(i) Transfer of data from master memory to slave memory.

(ii) Transfer of data from slave memory to master memory.

(iii) Master to initiate the execution of application program residing in slave memory.

4.2. Master Utility Programs

The master processor is equipped with software for control of all main system functions, such as, serial RS232C Communication between master and external Visual Display Terminal (VDU) and between master and an external host computer.

The system debugging utilities enable break-points to be inserted, registers of the master to be displayed and memory content to be examined and altered. The portion of the master software that controls data transfer between master and slave is listed in appendix 1.

4.3. Master Memory to Slave Memory Transfer

This transfer has been designated status code 2 (Fig. 4). Prior to any action, both status latches (master and slave) contain status code Ø, which is the null code, meaning that both processors are free to operate independently of each other.

The master initiates the transfer by writing hex 22 into the master status latch. This is code 2 and dataline d5 of the VME bus calls SPC interrupt level 4.

All memory to memory transfers require a destination address to precede the data. Hence the master passes to the slave two consecutive words which represent the 32 bit destination address of the data to be transferred. The slave automatically subtracts the offset from the address received (section 4.1) then incrementally stores the following data, starting at that address and continuing until the EOT, code 10 is received. The slave, on receipt of an EOT responds with code 0 (null), exits from the interrupt service routine and proceeds with the program it was executing prior to being interrupted by the master.

Two handshake codes (code 14 and code 15) are used to signify that data have been received and the processor is waiting for further response.

An outline of the mechanics of the master memory to slave memory transfer and the use of the status codes is shown in Fig. 8.

4.4. Slave Memory to Master Memory Transfer

This transfer has been designated Status Code 1 (Fig. 4). Again the action is initiated by the master, writing hex 21 into the master status latch. The process that then follows is similar to that described in Section 4.3 and detailed in Fig. 9.

4.5. Execution of Program with Slave Memory

Execution of application software residing in slave memory is designated status code 9 (Fig. 4). The start address for the execution of the application software is passed from the master to the slave and loaded into the slave program counter.

The slave commences execution of the previously stored software starting at that address.

No offset is involved with the execution address passed from the master to the slave. Fig. 10 details the process involved.

Appendix 2 provides a listing of the data transfer routines residing within the slave software. Complete listings are stored on the ARL Philips PMDS Microcomputer Development System under /arl-rout/cards/SPC.

5. CONCLUSION

The memorandum has presented a general description of both hardware and software utilized in this intelligent slave peripheral controller. For the application of positioning a six degree of freedom hot wire anemometer probe the controller has been successfully demonstrated using two high speed stepper motors.
(The other four motors are not yet available).

No difficulties were experienced in the control of two slaves by a single master. The full capabilities of the system have yet to be fully tested but indications are that the system is capable of control of stepper motors, using complex positioning algorithms, at speeds in excess of 10,000 steps/second. This controller could have application in any system which involves real time complex calculations for control, such as in high speed robotics.

The use of slave parallel 68000 microprocessors operating independently under the control of a master provides very powerful processing capabilities for high speed data acquisition and real time predictive control applications.

ACKNOWLEDGEMENTS

Of the many people who have made significant contributions to the development of this system, the author would particularly wish to record the contribution of W. Dekker and Dr J. Watmuff.

APPENDIX 1 DATA TRANSFER ROUTINES - MASTER SOFTWARE

```
!*************************************************************************
!       code 0  :: null each processor to do own thing
!       code 1  :: spc memory to vme memory transfer
!       code 2  :: vme memory to spc memory transfer
!       code 9  :: begin execution in spc memory : address from vme
!       code 10 :: eot end of transfer
!       code 14 and 15 :: ecode and fcode are data handshake codes
!       bit 5 of vme status calls interrupt level 4 spc
!        data address ffaf60 for chn1 , ffbf60 for chn2
!        status address ffaf51 for chn1 , ffbf51 for chn2
!
!*************************************************************************
!
!************   get start and finish address from keyboard **************
!                                          !start address in iopsad
!                                          !finish address in iopead
accadd: move.l  #ipiopm,a0
        jsr     txout.l
        jsr     space.l                   !o/p space to vdu
        jsr     fascii.l                  !get start address
        move.l  save.l,iopsad.l           !start in iopsad
        cmp.b   #0x1,d3                   !test if comma flag is set
        bne     xhot
        move.b  #0x2c,d0
        jsr     put.l                     !o/p comma to vdu
        jsr     fascii.l                  !get finish address
xhot:   move.l  save.l,iopead.l           !finish address
chnagn: jsr     crlf.l
        move.l  #meschn,a0                !message which spc 1 or 2
        jsr     txout.l
        jsr     get.l                     !get channel number
        jsr     put.l
        cmp.b   #'1',d0                   !test if channel 1
        beq     isone
        cmp.b   #'2',d0                   !test if channel 2
        bne     chnagn                    !not 1 or 2 try again
        move.l  #stats2,a5                !set up channel 2
        move.l  #datrg2,a6
        jsr     crlf.l
        rts
isone:  move.l  #stats1,a5                !set up channel 1
        move.l  #datrg1,a6
        jsr     crlf.l
        rts
```

```
!
!*********************  output address to spc  ***************************
!                                              ! enter with long address in d1
opsfad: swap    d1                             !change position of high & low words
        move.w  d1,(a6)                        !o/p high address word of data start
        move.b  #0x0e,(a5)                     !change handshake
        jsr     ecode                          !wait for ecode reply
        swap    d1                             !get low word back
        move.w  d1,(a6)                        !o/p low add word of data start
        move.b  #0x0f,(a5)                     !change handshake to fcode
        jsr     fcode                          !wait for fcode reply
        rts
!
!
!*****************************************************************************
!                  code 0 :: send and recieve null status
!*****************************************************************************
!
code0:  move.b  #0,(a5)                        !set vme status==null
await:  move.b  (a5),d0                        !get spc status
        and.l   #0x0f,d0                       !ensure only 4 last bits
        beq     acon                           !tests spc status is null
        bra     await                          !wait for spc status to become null
acon:   rts
!
!*****************************************************************************
!                 code A :: all data sent end transfer
!*****************************************************************************
!
acode:  move.b  #0x0a,(a5)                     !eot status code
bwait:  move.b  (a5),d0                        !handshake
        and     #0x0f,d0
        bne     bwait                          !if status==0 (null) then finished
        rts                                    !return from memory - memory transfer
!
!*****************************************************************************
!                   codes E & F :: handshake control codes
!*****************************************************************************
!
ecode:  move.b  (a5),d0                        !test status of vme
        and     #0x0f,d0
        cmp     #0x0e,d0                       !test if vme status==0e
        bne     ecode                          !not 0e so loop back and try again
        rts                                    !is 0e so return to main program
!
!
fcode:  move.b  (a5),d0                        !test status of vme
        and     #0x0f,d0
        cmp     #0x0f,d0                       !test if vme status==0f
        bne     fcode                          !not 0f so loop back and try again
        rts                                    !is 0f so return to main program
!
```

```
!
!*************************************************************************
!        code 1::  spc to vme  ::  memory to memory transfer
!*************************************************************************
!
code1:  jsr     code0.l                 !test if null are on status
        jsr     accadd.l                !get start and finish address
        move.l  iopsad.l,a4             !start address to a4
        add.l   #0x10000,a4             !offset on iopsad
        move.l  iopead.l,a3             !end address to a3
        add.l   #0x10000,a3             !offset on iopead
        move.b  #0x21,(a5)              !interrupt and code 1
cd1:    move.b  (a5),d0                 !wait for return of code 1
        and.l   #0x0f,d0
        cmp     #0x01,d0
        bne     cd1
        move.l  iopsad.l,d1
        jsr     opsfad.l                !o/p start address
dloop1: move.w  (a6),(a4)               !i/p data word into offset memory
        cmp.l   a4,a3                   !test for eot
        ble     recend                  !exit
        add     #2,a4                   !increment a4
        move.b  #0x0e,(a5)              !data recieved
        jsr     ecode                   !wait for data
        move.w  (a6),(a4)               !i/p data word
        cmp.l   a4,a3                   !test for eot
        ble     recend                  !exit
        add     #2,a4                   !increment a4
        move.b  #0x0f,(a5)              !data recieved
        jsr     fcode                   !wait for data
        bra     dloop1                  !loop until all data sent
!
!
recend: jsr     acode.l                 !eot to spc
        move.b  #0,(a5)                 !set null status
        move.l  iopsad.l,a1             !get start address
        add.l   #0x10000,a1             !add address offset
        move.l  a1,a1t.l
        add.l   #1,a4                   !last byte of last word
        move.l  a4,save.l               !get finish address
        clr.l   offlag.l
        clr.l   spnum.l
        bra     hot                     !o/p to vdu contents of iop memory
!
```

```
!
!*************************************************************************
!           code 2:: vme to spc ::  memory to memory transfer
!*************************************************************************
!
code2:  jsr     accadd.l                !get start and end address
        jsr     trmiop
        bra     red                     ! return to monitor
!
trmiop: jsr     code0.l                 !test null status
        move.l  iopsad,a4               !start address of data string
        sub.l   #0x10000,a4             !sub offset to start address
        move.l  iopead,a3               !finish address of data string
        sub.l   #0x10000,a3             !sub offset to finish address
        move.b  #0x22,(a5)              !interrupt and code2 to spc
cd2:    move.b  (a5),d0                 !test spc for code 2 status
        and     #0x0f,d0
        cmp     #0x02,d0
        bne     cd2                     !loop until code 2 sent by spc
        move.l  a4,d1                   !address to be o/p into d1
        jsr     opsfad.l                !o/p start address to spc
        add.l   #0x10000,a4
        add.l   #0x10000,a3
datiop: move.w  (a4),(a6)               !o/p data
        move.b  #0x0e,(a5)              !change handshake
        jsr     ecode                   !wait for handshake
        cmp.l   a4,a3                   !test end of memory transfer
        ble     endat                   !string complete if a4>a3
        add     #2,a4                   !increment a4
        move.w  (a4),(a6)               !o/p data word
        move.b  #0x0f,(a5)              !change handshake
        jsr     fcode                   !wait for handshake
        cmp.l   a4,a3                   !test end of memory transfer
        ble     endat                   !string complete if a4>a3
        add     #2,a4
        bra     datiop                  !loop until all data sent
!
!
endat:  jsr     acode.l                 !eot to iop
        move.b  #0,(a5)                 !set null status
        rts
```

```
!
!*************************************************************************
!              code 9:: execute program in spc memory
!*************************************************************************
!
!
code9:  move.l  #ipstm,a0               !message
        jsr     txout.l
        jsr     fascii.l                !get spc start address in d4
        jsr     chnagn                  !which spc
        jsr     cdex
        bra     red                     !return to monitor
!
cdex:   jsr     code0.l                 !test null status
        move.b  #0x29,(a5)              !interrupt and code 9 status
cd9:    move.b  (a5),d0                 !test for code 9
        and.l   #0x0f,d0
        cmp     #0x09,d0
        bne     cd9                     !wait for code 9 reply
        move.l  save.l,d4
        swap    d4                      !get high address first
        move.w  d4,(a6)                 !o/p high address
        move.b  #0x0e,(a5)              !change handshake to 0e
        jsr     ecode.l                 !wait for responding 0e
        swap    d4                      !get low address
        move.w  d4,(a6)                 !o/p low address
        jsr     acode.l                 !set eot and wait for spc null
        move.b  #0,(a5)                 !set vme status null
        rts
```

APPENDIX 2 DATA TRANSFER ROUTINES - SLAVE SOFTWARE

```
!*****************************************************************************
!
!       Activate interrupts and wait in loop for incoming command
!
!*****************************************************************************
!
!
!
iopst:  move.w  #0x2300,sr                      !set sr to allow interrurt
loop:   move.l  d0,d0                           !do nothing until
        bra     loop                            !interrupt level 4 auto vector
!
!
!********** interrupt level 4 auto vector (070) called  ***************
!
int4:   movem.l d0-d7/a0-a6,-(a7)               !save all registers
        move.b  status.l,d0                     !get code number
        and.b   #0x0f,d0
        cmp.b   #0,d0                           !test if null
        beq     nul
        cmp.b   #1,d0                           !test if code 1
        beq     code1
        cmp.b   #2,d0                           !test if code 2
        beq     code2
        cmp.b   #3,d0                           !test if code 3
        beq     code3
        cmp.b   #4,d0                           !test if code 4
        beq     code4
        cmp.b   #5,d0                           !test if code 5
        beq     code5
        cmp.b   #6,d0                           !test if code 6
        beq     code6
        cmp.b   #9,d0                           !test if code 9
        beq     code9
        cmp.b   #0x0a,d0                        !test if eot
        beq     eot
        bra     return                          !return code not recognised
!
!
return: movem.l (a7)+,a0-a6/d0-d7               !restore all registers
        bra     iopst                           !return from interrupt
!
comp:   move.b  #0,status.l                     !null status on spc
wait:   move.b  status.l,d0                     !waits for null on vme
        and     #0x0f,d0
        bne     wait
        bra     return
```

```
!
!*************** handshake control routines *****************
!
ecode:  move.b  status.l,dØ                     !get status
        and.b   #Øx Øf,dØ
        cmp.b   #Øx Øa,dØ                       !test eot code
        beq     comp                            !if eot exit
        cmp.b   #Øx Øe,dØ                       !test if vme status == Øe
        bne     ecode                           !not Øe, try again
        rts                                     !is ecode can return
!
fcode:  move.b  status.l,dØ                     !get status
        and.b   #Øx Øf,dØ
        cmp.b   #Øx Øa,dØ                       !test eot code
        beq     comp                            !if eot exit
        cmp.b   #Øx Øf,dØ                       !test if vme status == Øf
        bne     fcode                           !not Øf, try again
        rts                                     !is fcode can return
!
!*************************************************************
!          code 1  - iop memory to vme memory transfer       *
!*************************************************************
!
code1:  move.b  #1,status.l                     !respond with code 1
        jsr     ecode.l                         !wait for ecode
        move.w  datreg.l,dØ                     !read high word start address
        swap    dØ                              !put high word into top of dØ
        and.l   #Øxffff ØØØØ,dØ
        move.l  dØ,a4                           !high address into a4
        move.b  #Øx Øe,status.l                 !ecode onto status
        jsr     fcode.l                         !wait for responding fcode
        move.w  datreg.l,dØ                     !read low word start address
        and.l   #Øx ØØØØ ffff,dØ
        add.l   dØ,a4                           !complete address in a4
nxdat:  move.w  (a4)+,datreg.l                  !o/p data from memory to vme
        move.b  #Øx Øf,status.l
        jsr     ecode.l
        move.w  (a4)+,datreg.l                  !o/p data to vme
        move.b  #Øx Øe,status.l
        jsr     fcode
        bra     nxdat                           !loop until data transferred
!
```

```
!
!*****************************************************************
!          code 2  - vme memory to spc memory transfer           *
!*****************************************************************
!
code2:  move.b  #2,status.l                     !respond with code 2
        jsr     ecode.l                         !wait for ecode
        move.w  datreg.l,d0                     !reads high word start address
        swap    d0                              !puts high word into top of d0
        and.l   #0xffff0000,d0
        move.l  d0,a4                           !high address into top of a4
        move.b  #0x0e,status.l                  !puts ecode into status
        jsr     fcode.l                         !wait for responding fcode
        move.w  datreg.l,d0                     !reads low word start address
        and.l   #0x0000ffff,d0
        add.l   d0,a4                           !complete address into a4
        move.b  #0x0f,status.l                  !puts fcode into status
datlp:  jsr     ecode.l                         !wait for ecode or eot reply
        move.w  datreg.l,(a4)+                  !read data and store in memory
        move.b  #0x0e,status.l                  !puts ecode into status
        jsr     fcode.l                         !waits for fcode or eot reply
        move.w  datreg.l,(a4)+                  !read data and store in memory
        move.b  #0x0f,status.l                  !puts fcode into status
        bra     datlp                           !loop until all data received
!
!
!*****************************************************************
!          code 9  execute a program in spc memory               *
!*****************************************************************
! address high - address low > spc to begin execution at.
!
code9:  move.b  #9,status.l                     !respond with code 9
        jsr     ecode                           !wait for handshake
        move.w  datreg.l,d0                     !read high address
        swap    d0                              !high address into top of d0
        and.l   #0xffff0000,d0
        move.l  d0,a0
        move.b  #0x0e,status.l                  !handshake
cd9:    move.b  status.l,d0                     !read status
        and.b   #0x0f,d0
        cmp.b   #0x0a,d0
        bne     cd9                             !wait for code A
        move.w  datreg.l,d0                     !read low address
        and.l   #0x0000ffff,d0
        add.l   d0,a0                           !complete address in a0
        move.b  #0,status.l                     !null to status
        jmp     (a0)                            !begin execution at (a0)
!
!
```

Table 1. ARL Drawing Numbers for SPC Hardware

ARL Drawing Number	Description
60278-A2	Parts List - Hardware
60279-A2	General Assembly - Hardware
60280-F3	Art Work - Front Panel
60281-A2	Parts List - Main Card (CPU)
60282-A1	Circuit Diagram - Main Card (CPU)
60283-A2	Assembly - Main Card (CPU)
60284-F2	Art Work 1 - 2:1 Lay up Main Card (CPU)
60285-F2	Art Work 2 - 2:1 Lay Up Main Card (CPU)
60286-A2	Parts List - Piggy-back Card (Memory)
60287-A1	Circuit Diagram - Piggy-back Card (Memory)
60288-A2	Assembly - Piggy-back card (Memory)
60289-F2	Art Work 1 - 2:1 Lay up Piggy-back Card (Memory)
60290-F2	Art Work 2 - 2:1 Lay up Piggy-back Card (Memory)
60330-A3	Detail - Front Panel Cut-Out

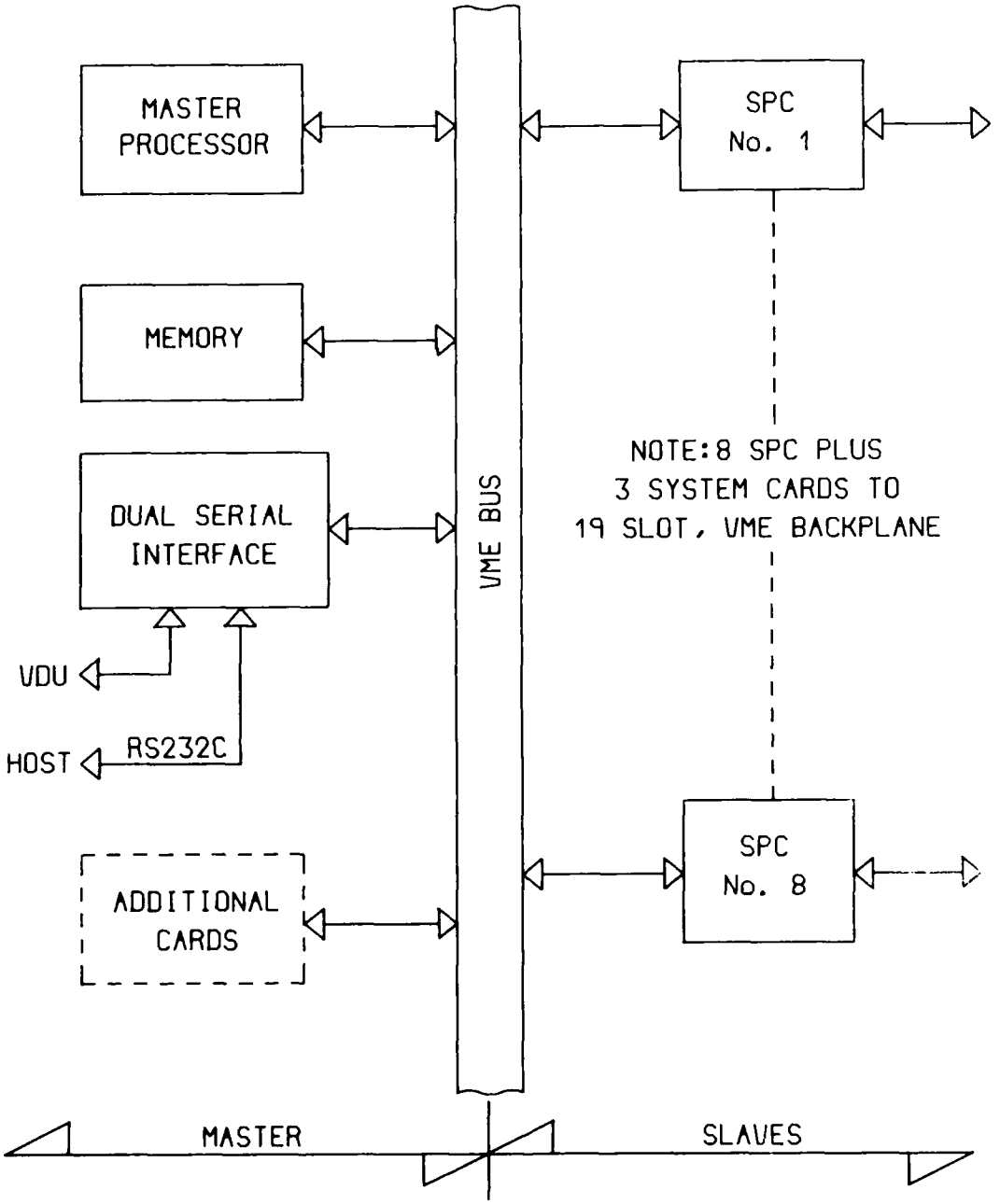


FIG. 1 MASTER – SLAVE SYSTEM CONFIGURATION

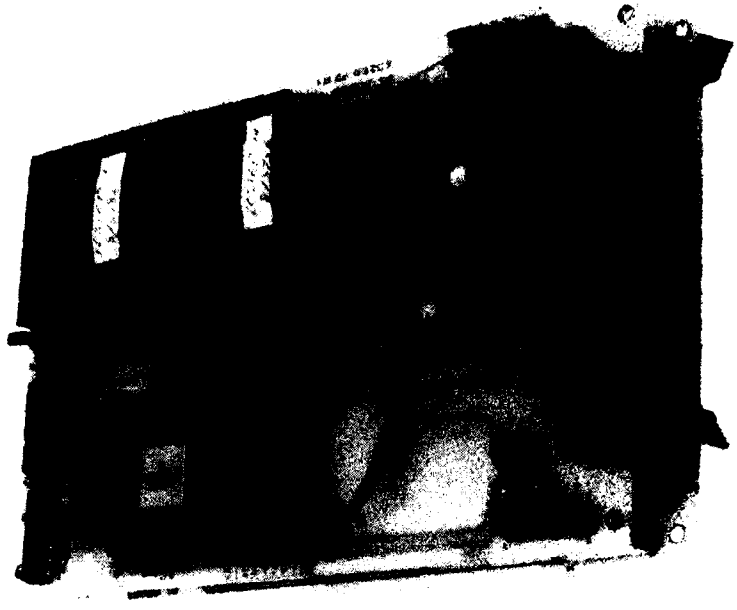


FIG. 2(a) PIGGY-BACKED SPC

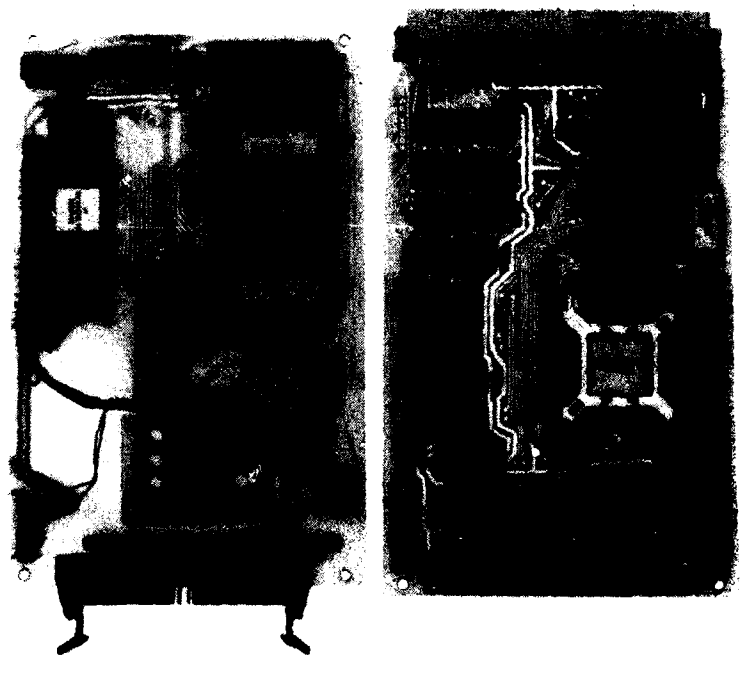


FIG. 2(b) SEPARATED SPC

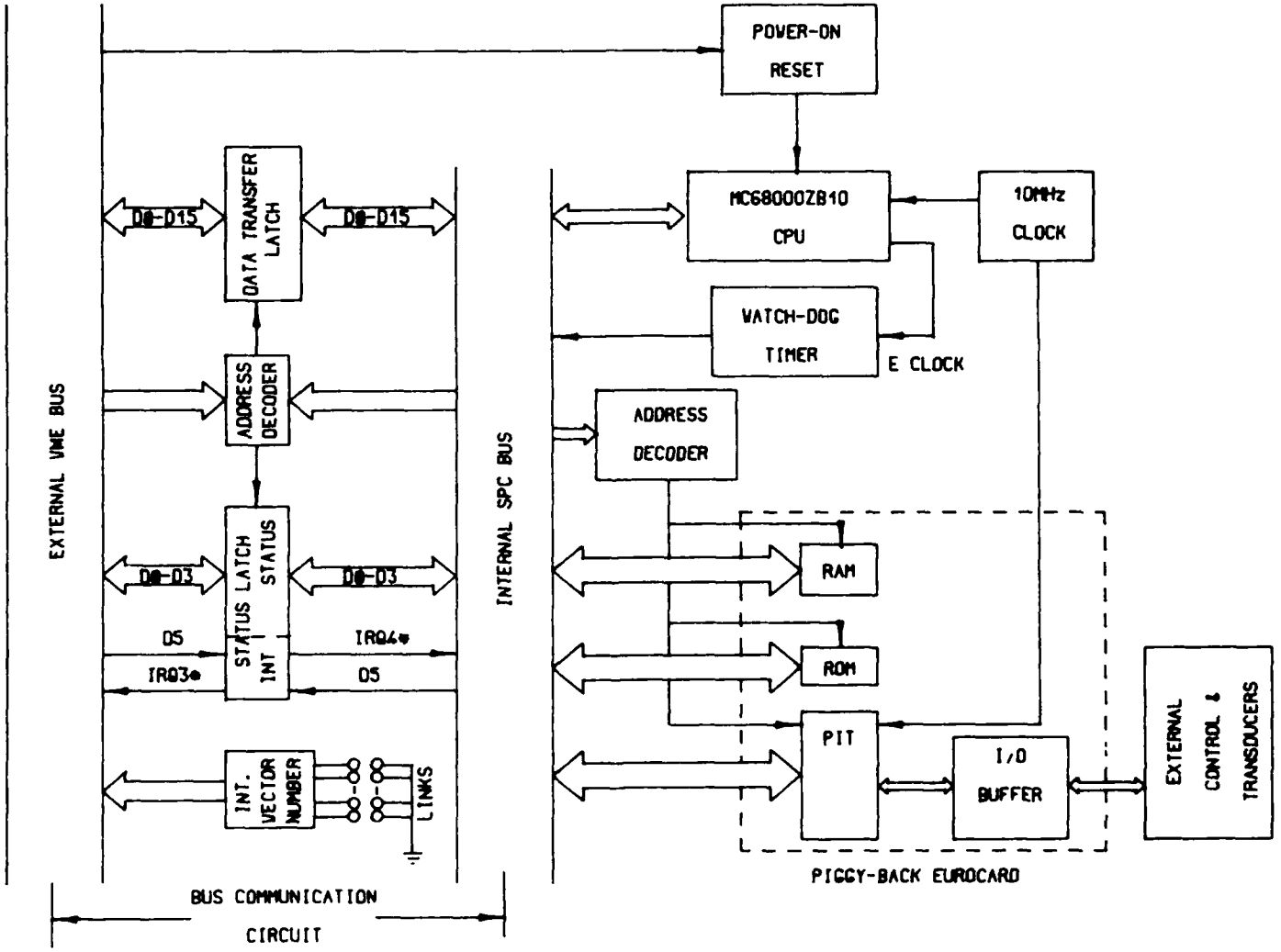


FIG. 3 SLAVE PERIPHERAL CONTROLLER (SPC) – SCHEMATIC

CODE	HEX	OPERATION
0	00	NULL
1	01	SPC Memory to VME Memory
2	02	VME Memory to SPC Memory
3	03	
4	04	
5	05	
6	06	
7	07	
8	08	
9	09	Execute program in SPC Memory
10	0A	End of transmission EOT
11	0B	
12	0C	
13	0D	
14	0E	Handshake e code
15	0F	Handshake f code

FIG. 4 SPC STATUS CODES

Vector Number(s)	Address			Assighment
	Dec	Hex	Space	
0	0	000	SP	Reset: Initial SSP
–	4	004	SP	Reset: Initial PC
2	8	008	SD	Bus Error
3	12	00C	SD	Address Error
4	16	010	SD	Illegal Instruction
5	20	014	SD	Zero Divide
6	24	018	SD	CHK Instruction
7	28	01C	SD	TRAPV Instruction
8	32	020	SD	Privilege Violation
9	36	024	SD	Trace
10	40	028	SD	Line 1010 Emulator
11	44	02C	SD	Line 1111 Emulator
12	48	030	SD	(Unassigned, reserved)
13	52	034	SD	(Unassigned, reserved)
14	56	038	SD	(Unassigned, reserved)
15	60	03C	SD	Uninitialized Interrupt Vector
16-23	64	04C	SD	(Unassigned, reserved)
	95	05F		–
24	96	060	SD	Spurious Interrupt
25	100	064	SD	Level 1 Interrupt Autovector
26	104	068	SD	Level 2 Interrupt Autovector
27	108	06C	SD	Level 3 Interrupt Autovector
28	112	070	SD	Level 4 Interrupt Autovector
29	116	074	SD	Level 5 Interrupt Autovector
30	120	078	SD	Level 6 Interrupt Autovector
31	124	07C	SD	Level 7 Interrupt Autovector
32-47	128	080	SD	TRAP Instruction Vectors
	191	0BF		–
48-63*	192	0C0	SD	(Unassigned, reserved)
	255	0FF		–
64-255	256	100	SD	User Interrupt Vectors
	1023	3FF		–

FIG. 5 MOTOROLA MC 68000 EXECPTION VECTOR ASSIGNMENT

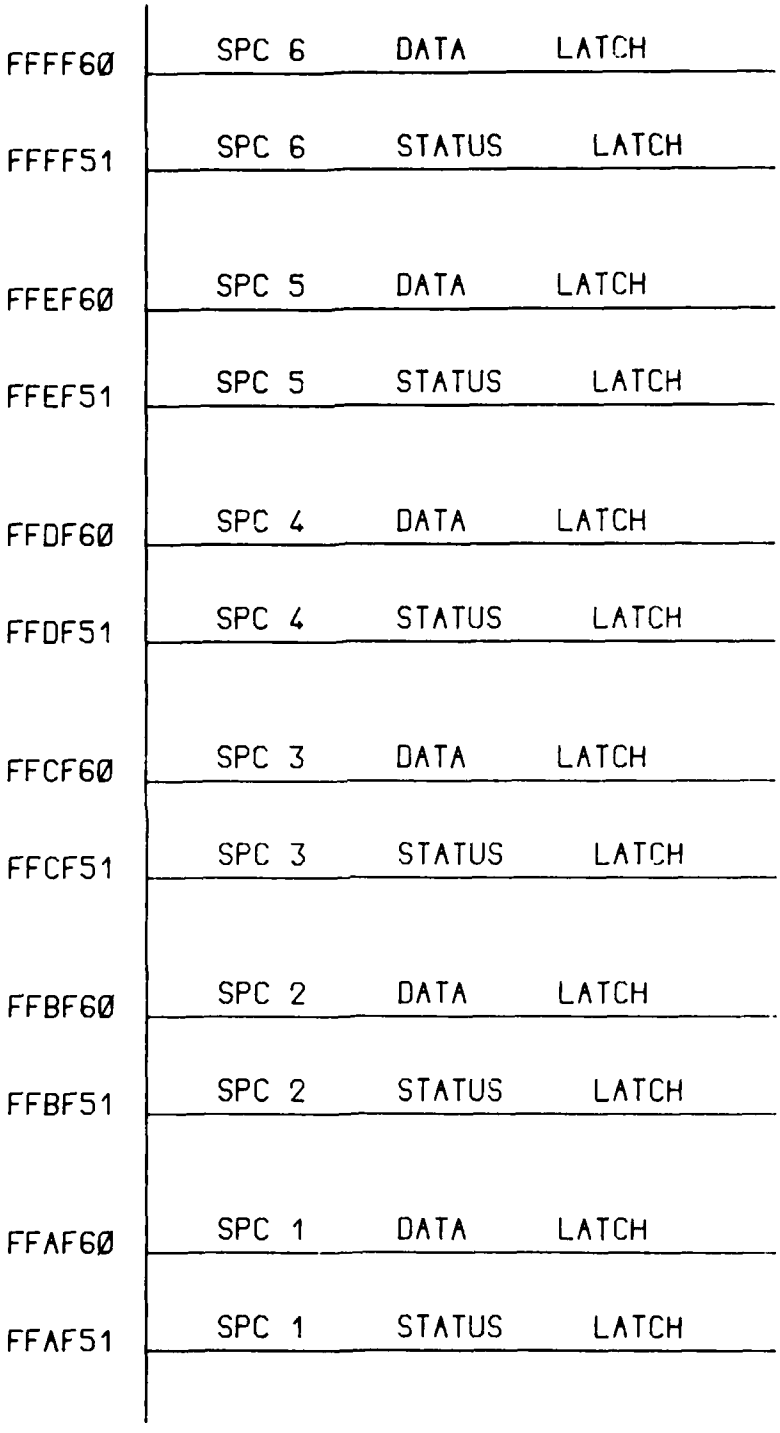


FIG. 6 VME BUS ADDRESS MAP FOR SIX SPC'S

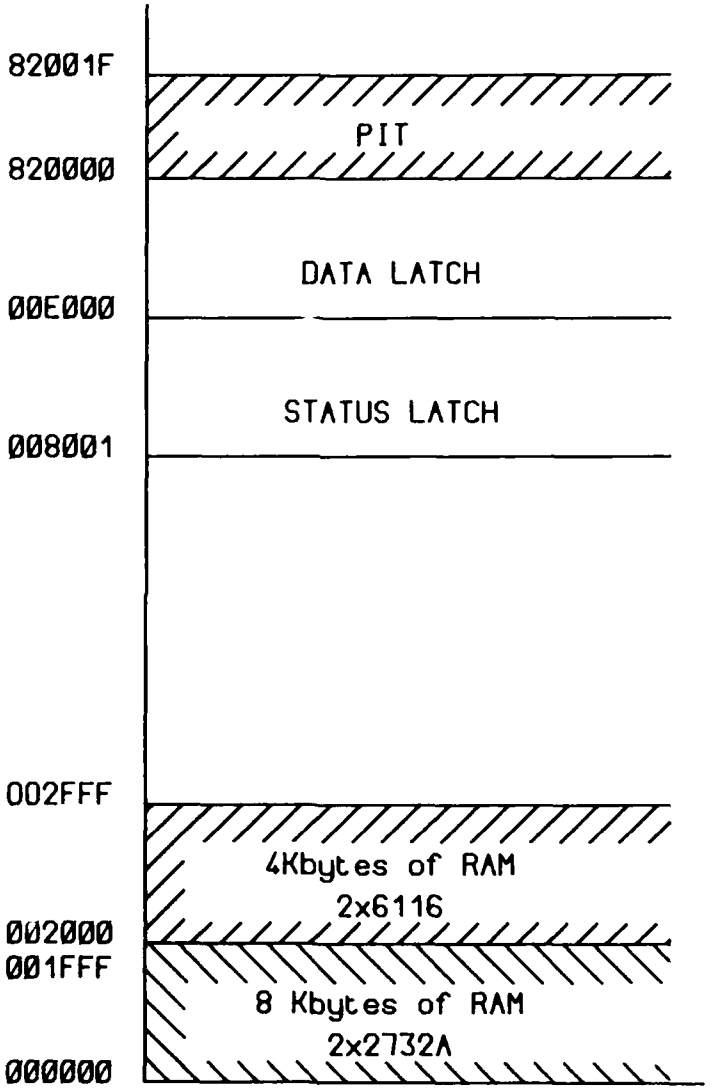


FIG. 7 SPC ADDRESS MAP

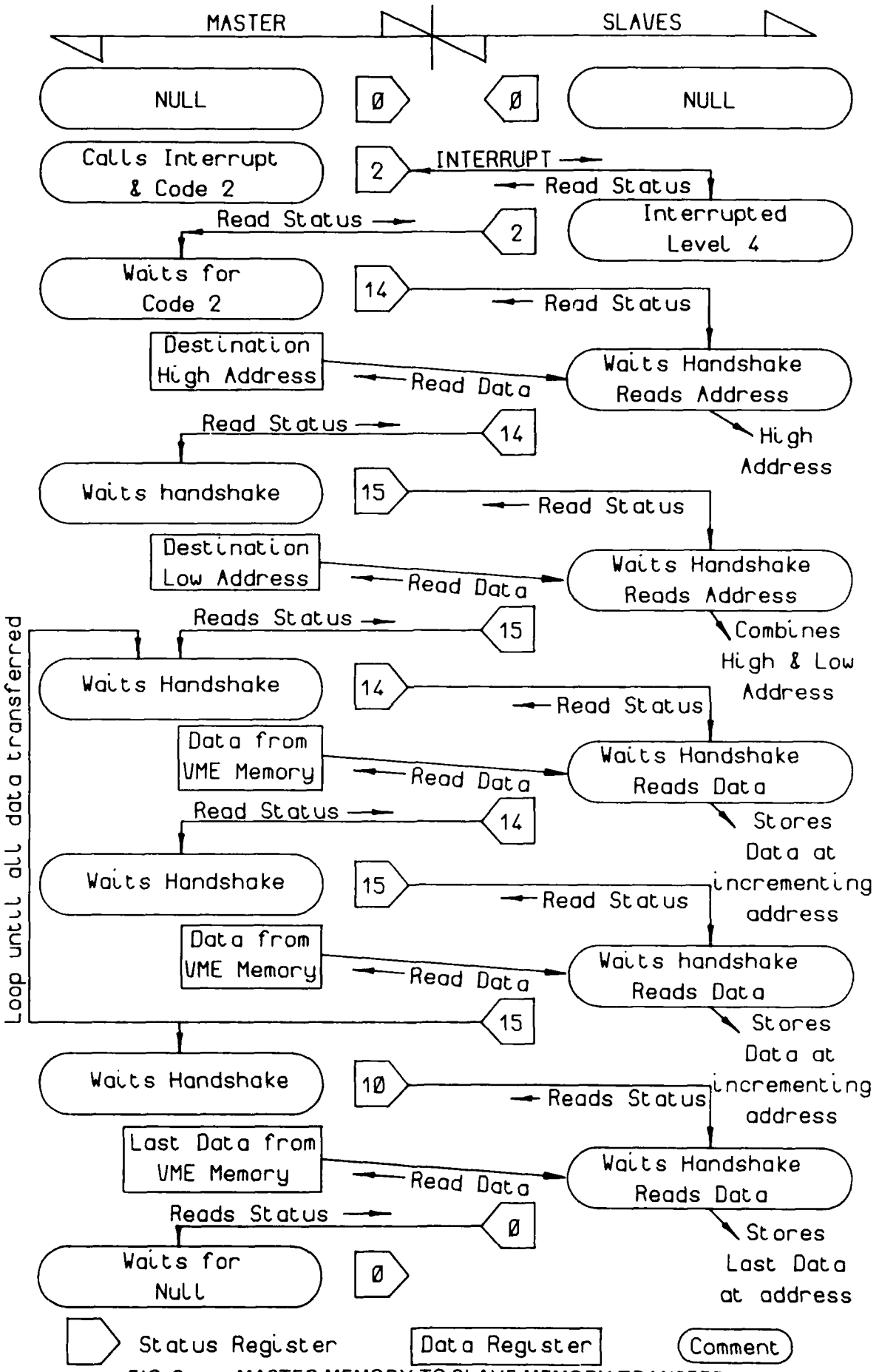


FIG. 8 MASTER MEMORY TO SLAVE MEMORY TRANSFER

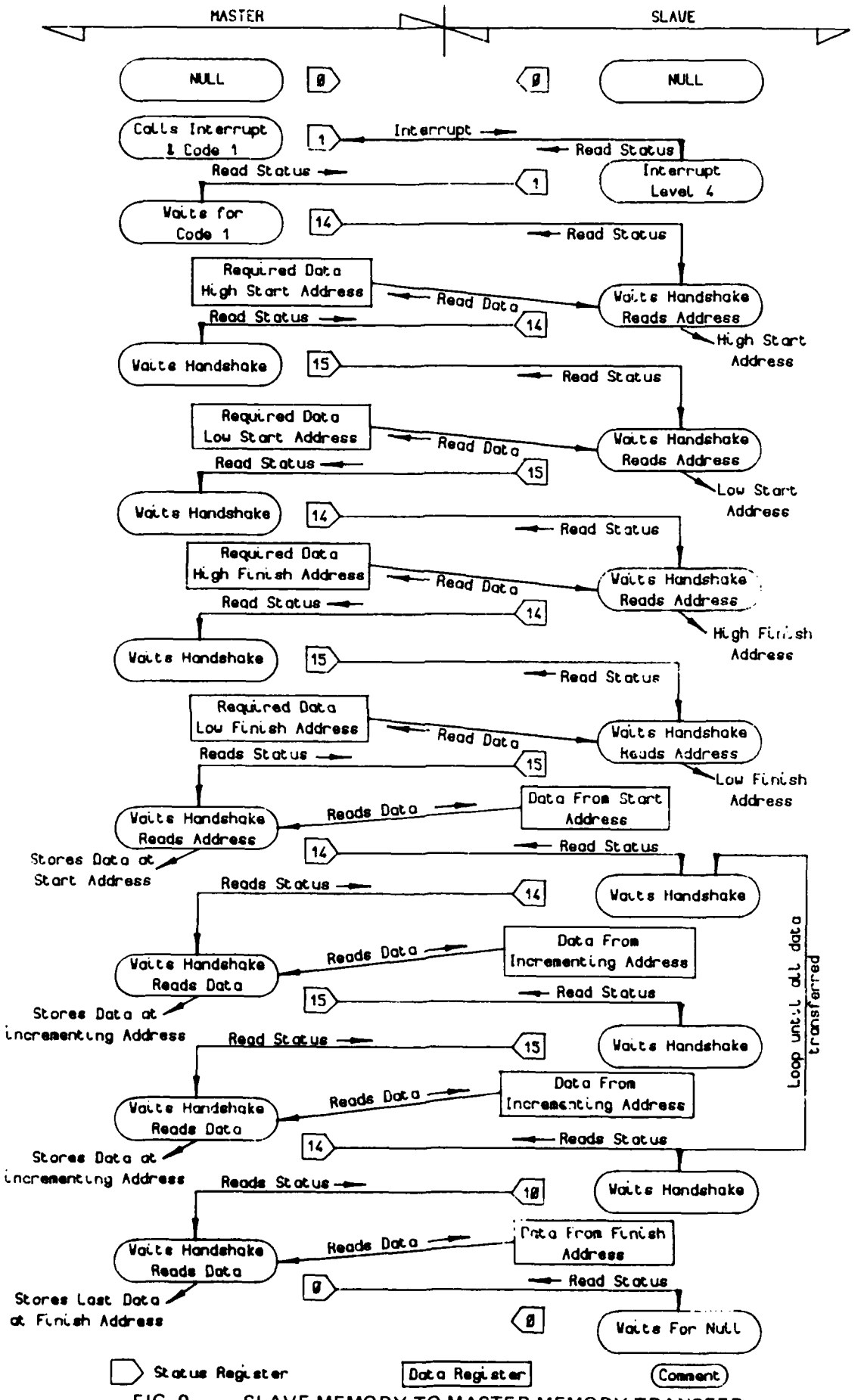


FIG. 9 SLAVE MEMORY TO MASTER MEMORY TRANSFER

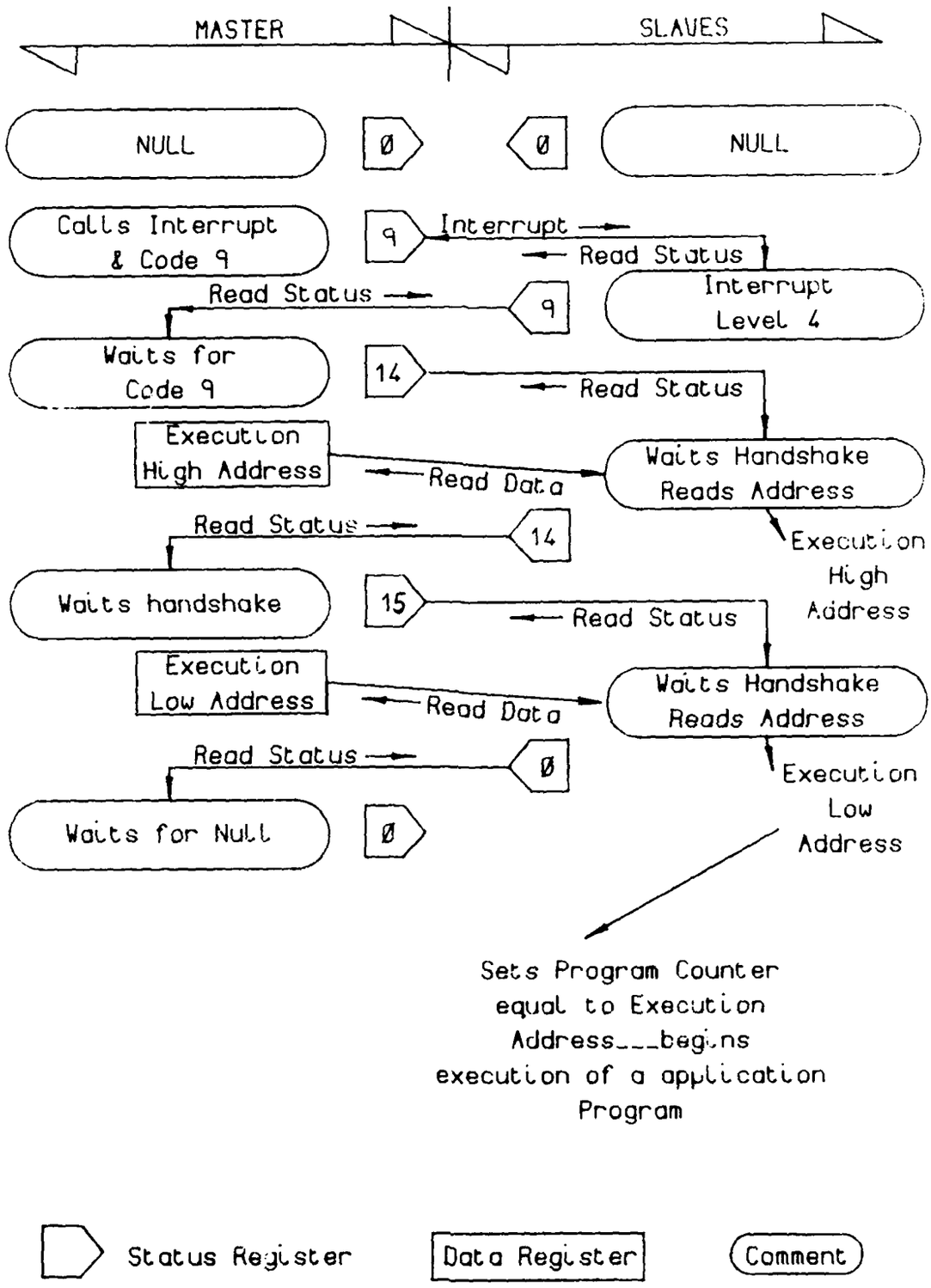


FIG. 10 EXECUTION OF PROGRAM IN SPC MEMORY

DISTRIBUTION

AUSTRALIA

Department of Defence

Defence Central
- Chief Defence Scientist
- Deputy Chief Defence Scientist (shared copy)
- Superintendent, Science and Program Administration (shared copy)
- Controller, External Relations, Projects and Analytical Studies (shared copy)
- Counsellor, Defence Science (London) (Doc Data Sheet Only)
- Counsellor, Defence Science (Washington) (Doc Data Sheet Only)
- Defence Science Representative (Bangkok)
- Defence Central Library
- Document Exchange Centre, DISB (18 copies)
- Joint Intelligence Organisation
- Librarian H Block, Victoria Barracks, Melbourne
- Director General - Army Development (NSO) (4 copies)

Aeronautical Research Laboratories
- Director
- Library
- Superintendent - Aerodynamics
- Divisional File - Aerodynamics
- Group File (IFEG) - Aerodynamics (2 copies)
- N. Pollock
- B. Fairlie
- Author: J.F. Harvey

Materials Research Laboratories
- Director/Library

Defence Research Centre
- Library

SPARES (3 copies)
TOTAL (41 copies)

Department of Defence

DOCUMENT CONTROL DATA

1.a. AR No	1.b. Establishment No	2. Document Date	3. Task No
AR-004-507	ARL-AERO-TM-385	NOVEMBER 1986	DST 82/021

4. Title	5. Security	6. No Pages
A MULTI-PROCESSOR SLAVE PERIPHERAL CONTROLLER	a. document: UNCLASSIFIED b. title: U c. abstract: U	16 7. No Refs: 3

8. Author(s)	9. Downgrading Instructions
J.F. HARVEY	

10. Corporate Author and Address	11. Authority (as appropriate) a.Sponsor b.Security c.Downgrading d.Approval
Aeronautical Research Laboratories P.O. Box 4331, MELBOURNE, VIC. 3001	

12. Secondary Distribution (of this document)

Approved for public release.

Overseas enquirers outside stated limitations should be referred through ASDIS, Defence Information Services Branch, Department of Defence, Campbell Park, CANBERRA ACT 2601

13.a. This document may be ANNOUNCED in catalogues and awareness services available to ...

No limitations.

13.b. Citation for other purposes (ie casual announcement) may be [redacted] unrestricted [redacted].

14. Descriptions	15. COSATI Group
Controllers Multiprocessors Robots Data processing	09030

16. Abstract

A system providing multiple slave co-processors operating independently, yet in parallel upon the same VME bus, all under the control of a master processor is described.

Data are passed unidirectionally from master to slave enabling rapid processing of complex algorithms for the control of external peripherals.

PF 65

This paper is to be used to record information which is required by the Establishment for its own use but which will not be added to the DISTIS data base unless specifically requested.

16. Abstract (contd)

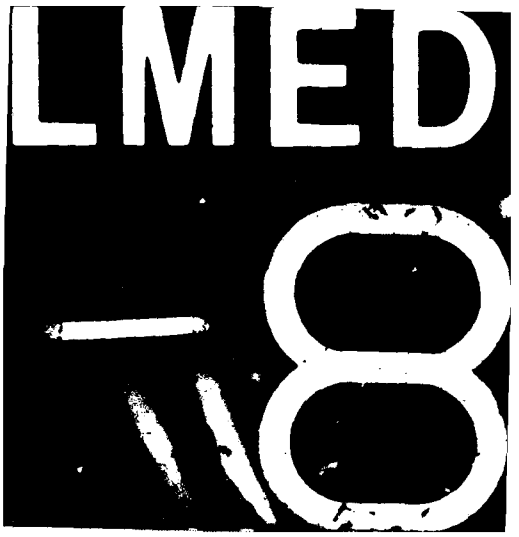
17. Imprint

Aeronautical Research Laboratories, Melbourne

18. Document Series and Number	19. Cost Code	20. Type of Report and Period Covered
Aerodynamics Technical Memorandum 385	55 6055	

21. Computer Programs Used

22. Establishment File Ref(s)


LMED
8